CS304 Practice midterm

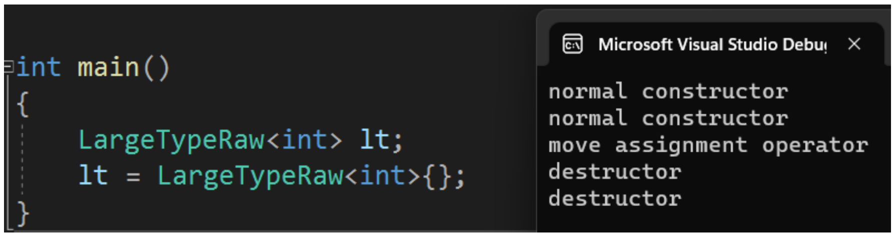
The practice midterm is longer and has more difficult questions than the real midterm (don't freak out).

Q1 write the copy constructor and move constructor for the following class

```
class LargeTypeRaw
{
public:
	LargeTypeRaw(int initialSize=10):
		size{initialSize},data{new int[initialSize]}
	{}

	// a) copy constructor here

	// b) move constructor here

private:
	int* data;
	int size;
};
```

Q2 assume LargeTypeRaw is a class similar to what we did in assignment 1, with print statements in the big 5. What will the following programs print (an example is given below)?

Example program with output on right.



```
int main()
{
    LargeTypeRaw<int> lt;
    lt = LargeTypeRaw<int>{};
}
```

Microsoft Visual Studio Debu

```
normal constructor
normal constructor
move assignment operator
destructor
destructor
```

Program 1: (show the output)

```
int main()
{
    LargeTypeRaw<int> lt;
    LargeTypeRaw<int> lt2 = lt;
}
```

Program 2: (show the output)

```
int main()
{
    std::vector<LargeTypeRaw<int>> largeTypes(2);
    largeTypes[0] = LargeTypeRaw<int>{};
    largeTypes[1] = largeTypes[0];
}
```

Program 3: (show the output)

```
int main()
{
    std::vector<LargeTypeRaw<int>> largeTypes;
    largeTypes.push_back(LargeTypeRaw<int>{});
    largeTypes[0] = LargeTypeRaw<int>{};
}
```

Q3 what is the time complexity of the following operations on a std::list

a) push_front(Object elem) // add element to front of list
b) pop_front() // remove element from front of list
c) insert(iterator position, Object elem) // insert element after position in list
d) remove(Object elem) // find and remove element

Q4 what is the time complexity of the following operations on a std::vector

a) vec[i] // access an element in the vector
b) erase(iterator first, iterator last) // erase elements between first and last
c) push_back(Object elem) // add an object to end of vector

Q5 write the code to remove duplicates from a sorted linked list, assuming the following 'Node' definition:

```
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     ListNode *next;
 *     ListNode() : val(0), next(nullptr) {}
 *     ListNode(int x) : val(x), next(nullptr) {}
 *     ListNode(int x, ListNode *next) : val(x), next(next) {}
 * };
 */
class Solution {
public:
    ListNode* deleteDuplicates(ListNode* head) {

    }
};
```

Q6 insertion sort

a) convert the following pseudocode to c++

```
INSERTION-SORT(A)
1  for $j \leftarrow 2$ to $length[A]$
2      do $key \leftarrow A[j]$
3         $\triangleright$ Insert $A[j]$ into the sorted sequence $A[1 \,..\, j-1]$.
4         $i \leftarrow j - 1$
5         while $i > 0$ and $A[i] > key$
6             do $A[i+1] \leftarrow A[i]$
7                $i \leftarrow i - 1$
8         $A[i+1] \leftarrow key$
```

b) show the state of the array a=[6,5,4,3,2,1] after the first 3 iterations of the outer loop (j index) of your c++ code

Q7 what will the following program print when it runs?

```
void merge(std::vector<int>& a, int p, int q, int r)
{
	std::cout << "merging, p=" << p << " q=" << q << " r=" << r << std::endl;
	// code to merge below, not needed to understand the output
}

void mergeSort(std::vector<int>& a, int p, int r)
{
	if (p < r)
	{
		int q = (p + r) / 2;
		mergeSort(a, p, q);
		mergeSort(a, q + 1, r);
		merge(a, p, q, r);
	}
}

int main()
{
	std::vector<int> list = {3,2,1};
	mergeSort(list,0,list.size()-1);
}
```

Q8 what is the best-case and worst-case time complexity of the following sorting algorithms

a) Mergesort
b) Quicksort
c) Insertion sort

Q9 evaluate the following postfix expressions:

12+

34*24*+

Q10 convert the following infix expressions to postfix:

(a+b+c)*e

1+2+3+4+5+6*7

Q11 what ADT does the following description correspond to (I replaced the data structure name with blanks) :

______ *are sequence containers representing arrays that can change in size.*
Just like arrays, ______ *use contiguous storage locations for their elements, which means that their elements can also be accessed using offsets on regular pointers to its elements, and just as efficiently as in arrays. But unlike arrays, their size can change dynamically, with their storage being handled automatically by the container.*

Internally, _____ use a dynamically allocated array to store their elements. This array may need to be reallocated in order to grow in size when new elements are inserted, which implies allocating a new array and moving all elements to it. This is a relatively expensive task in terms of processing time, and thus, vectors do not reallocate each time an element is added to the container.

Q12 what ADT does the following description correspond to (I replaced the data structure name with blanks) :

_____ are sequence containers that allow constant time insert and erase operations anywhere within the sequence.
_____ are implemented as singly-linked lists; Singly linked lists can store each of the elements they contain in different and unrelated storage locations. The ordering is kept by the association to each element of a link to the next element in the sequence.
The main design difference between a _____ container and a list container is that the first keeps internally only a link to the next element, while the latter keeps two links per element: one pointing to the next element and one to the preceding one, allowing efficient iteration in both directions, but consuming additional storage per element and with a slight higher time overhead inserting and removing elements. _____ objects are thus more efficient than list objects, although they can only be iterated forwards.

Q13 is a stack an abstract data type or a data structure? Justify your response.

Q14 describe in your own words why shell sort is faster than insertion sort.

Q15 which data structure is more appropriate for representing a digital image (photograph): a) linked list or b) array?

Bonus: what is the time complexity of bogo sort? Is there an input for which bogo sort will outperform quicksort?